EYouth X DEPI Tech Challenge

Submitted by:
EYouth

Tic-Tac-Toe(XO) Game

Challenge Description:

Develop a Tic-Tac-Toe (XO) Web App using ASP.NET Core. Students will create a functional game where a player can compete either against another player or an AI opponent, implementing game logic and user input handling.

Implementation in Steps:

- Step 1: Setup: Create an ASP.NET Core Web App using MVC.
- Step 2: Game Logic: Handle turns, win conditions, and game state using Database SQL Server.
- Step 3: Frontend: Build an interactive UI with HTML, CSS, and JavaScript.
- Step 4: Game Modes: Allow single-player (against Web Server) and multiplayer (against another player) options.
- Step 5: Multilayer (Optional): Use Clean Architecture or onion architecture.
- Step 6: Testing & Deployment: Ensure functionality and provide a working demo.

Supporting Material:

ASP.NET Core Docs: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

Basic XO Game Logic in C#: Just Hint:

"

```
public class TicTacToe
{
    private char[,] board = new char[3, 3];
```

```
    private char currentPlayer = 'X';

    public void MakeMove(int row, int col)

    {

        if (board[row, col] == '\0')

        {

            board[row, col] = currentPlayer;

            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

        }

    }

}
```

"

Time Frame:

- Duration: 1 Week
- Deadline: 28/2/2025.

Student Deliverables:

- Source Code: GitHub repository or ZIP file.
- Demo Video (Optional): Brief explanation of the game.
- Documentation: README file with project structure and setup steps.
- Live Deployment (Bonus): Hosting on a Free Server like Smarter or any other free server.

Evaluation criteria:

1. **Project Setup and Structure (10%)**

- Correct Framework: The project is set up using ASP.NET Core MVC as specified.
- Project Structure: The project follows a clean and organized structure, with proper separation of concerns (e.g., Models, Views, Controllers).
- README File: A comprehensive README file is provided, explaining the project structure, setup steps, and how to run the application.

2. **Game Logic Implementation (30%)**

- **Turn Handling**: The game correctly handles player turns, alternating between 'X' and 'O'.
- **Win Conditions**: The game accurately detects win conditions (e.g., three in a row, column, or diagonal).
- **Draw Condition**: The game correctly identifies a draw when the board is full with no winner.
- **Database Integration**: The game state is stored and managed using **SQL Server** as specified.
- **Error Handling**: The game handles invalid moves gracefully (e.g., preventing overwriting existing moves).

3. **Frontend Development (20%)**

- **Interactive UI**: The frontend is interactive, allowing users to click on cells to make moves.
- **Responsive Design**: The UI is responsive and works well on different screen sizes (desktop, tablet, mobile).
- **Visual Appeal**: The design is visually appealing, with clear indications of 'X' and 'O' and game state (e.g., win/draw messages).
- **JavaScript Usage**: JavaScript is used effectively to handle user interactions and update the UI dynamically.

4. Game Modes (20%)

- **Single-Player Mode**: The game allows a player to compete against an AI opponent.
- **Multiplayer Mode**: The game supports two players competing against each other.
- **AI Implementation (Optional)**: If implemented, the AI opponent makes logical moves (e.g., random or basic strategy).
- **Mode Switching**: The game allows users to switch between single-player and multiplayer modes easily.

5. Testing and Deployment (10%)

- **Functionality Testing**: The game is thoroughly tested, with no critical bugs or issues.
- **Deployment**: The game is deployed to a live server (e.g., SmarterASP.NET or any other free server) and is accessible via a URL.